Modbus Gateway User Manual (3.30 Release 3 Firmware)

Copyright © **Delta Controls Inc.** All rights reserved

Document Title: Modbus Gateway User Manual
Document Number: Not required, released as PDF file
Current Revision: Edition 2.1
Date of current revision: October 5, 2005

No part of this manual may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language (natural or computer), in any form or by any means, without the prior written permission of Delta Controls Inc.

Limited permission is granted to reproduce documents released in Adobe® Portable Document Format (PDF) electronic format in paper format. Documents released in PDF electronic format may be printed by end users for their own use using a printer such as an inkjet or laser device. Authorized distributors of Delta Controls Inc. products (Delta Partners) may print PDF documents for their own internal use or for use by their customers. Authorized Delta Partners may produce copies of released PDF documents with the prior written permission of Delta Controls Inc.

Information in this document is subject to change without notice and does not represent a commitment to past versions of this document on the part of Delta Controls Inc. Delta Controls Inc. may make improvements and/or changes to this manual/the associated software/or associated hardware at any time.

Delta Controls Inc. assumes no responsibility for any errors that may appear in this document or any damages and/or losses resulting from the use of the product.

Note that Delta software programs described in this manual are copyrighted material, and when you purchase any of the software programs you are granted a limited license to use the software as described in this and other manuals.

BACspec, BACstat, Delta Controls, ORCAview, and Virtual Stat are trademarks of Delta Controls Inc.
Windows is a registered trademark of Microsoft Corporation.
Adobe and Acrobat are registered trademarks of Adobe Systems Incorporated.
BACnet is a registered tradmark of the American Society of Heating, Refrigeration and Air-Conditioning Engineers, Inc. (ASHRAE).
BTL is a registered trademark of the BMA
Modbus is a trademark of Schneider Automation.
All other trademarks are the property of their respective owners.

Table Of Contents

TABLE OF CONTENTS **3**

USING THIS MANUAL **4**

ORGANIZATION OF THIS DOCUMENT 4
OTHER RELEVANT DOCUMENTS 4

INTRODUCTION **5**

IMPORTANT INFORMATION **5**

AVAILABLE MODBUS PRODUCTS **5**

MODBUS GATEWAYS 5
MODBUS CONTROLLERS 6
FLASH LOADING MODBUS UNITS INTO A DEVICE 6
FLASH LOADER KEY & CREDITS 7

CONNECTING TO THE MODBUS DEVICE **7**

CONFIGURING THE MODBUS GATEWAY **9**

COMMUNICATIONS SETTINGS 9
LINKING MODBUS DEVICE REGISTERS TO BACNET OBJECTS 10
GWT CONFIGURATION EXAMPLES 11
Reading Input Status Registers 11
Reading/Writing Coil Registers 12
Reading Input Registers 12
Reading/Writing Holding Registers 12
MEMORY LIMITS & PERFORMANCE ISSUES 13

APPENDICES **14**

APPENDIX A: TROUBLESHOOTING 14
Gateway Object Statistics Tab 14
Fault Flag Linked to a Modbus Register 14
Exception Code Responses 15
Tx & Rx Packet Snooping 15
APPENDIX B: GW AND GWT OBJECT REFERENCE 16
Gateway Object (GW) 16
Gateway Translation Object (GWT) 20
APPENDIX C: MODBUS PROTOCOL IMPLEMENTATION 25
Memory Types 26
Data Types 27
Functions 30
Exception Responses 37

DOCUMENT CONTROL **38**

Using this Manual

This manual is designed to inform the user of the specific features unique to Delta's Modbus Gateway controllers. Users should have previous training on Delta Controls products. Specifically the user will need to know how to use ORCAview to connect to a controller, load and save databases, open and create objects in navigator, and enter data using object dialogs.

Organization of this Document

This document explains how to configure a Modbus Gateway Special Interface controller to communicate with one or more Modbus slave devices using Delta's ORCAview Operator Workstation software package.

The document describes:

- Software Communications setup
- Linking Modbus Registers to BACnet Objects
- Appendices: Troubleshooting, GW & GWT Object Reference, Modbus Protocol Implementation

Other Relevant Documents

This document only contains part of the information required to correctly install and program a Modbus Gateway Special interface controller to communicate with one or more Modbus devices. Refer to the documents listed below for supplementary information:

- **Installation Guides:** For installation instructions refer to the install guide for the standard version of the associated Modbus Gateway controller. For example installation instructions for a DSC-1212-MOD5 Modbus Gateway controller can be found in the DSC-1212 Installation Guide.
- **Delta Controls Wiring Guidelines:** This document contains instructions on recommended practices when wiring power, communications and IO to Delta controllers.
- **ORCAview Technical Reference Manual:** For general database configuration and programming instructions.
- **3.30 Release 3 Modbus Gateway Products Release Notes:** For a summary of new features and any known issues with this version of the Delta Modbus Gateway firmware.
- **Modbus Slave Device Documentation:** In order to configure the Modbus Gateway controller to communicate with a Modbus slave device you need to know the devices required communications type (RS-485 or RS-232), network address and speed. As well as the Register mappings including data types (Float, Integer etc.)
- **Modicon Modbus Protocol Reference Guide:** This document provides a detailed description of the Modbus protocol specification.

Introduction

Many of Delta's Controllers have an ordering option that allows them to function as a Modbus master device and communicate with one or more Modbus slave devices. This allows the controller to act as a Gateway translating information to/from the Modbus slave device(s) into BACnet objects resident in the Gateway controller, which can then be accessed by the rest of the BACnet network.

The Delta Modbus Gateway controllers communicate to Modbus slave devices using the Modbus RTU protocol via its' NET (RS-485) or PTP (RS-232) ports.

Important Information

When entering data into the Gateway Translation object (GWT) dataview do not use the insert or delete keys to insert or delete an entry in the dataview. There is currently a bug with this functionality in the GWT object that will result in the object being corrupted. This will be fixed in a future revision of the Modbus Gateway firmware.

Available Modbus Products

Modbus Gateways

Gateway products are intended to be used where a gateway to BACnet is required for *a number of Modbus devices (or units) communicating on the same Modbus network* (such providing BACnet access to a bunch of Rooftop units connected together on a Modbus network. These are the only products that come pre-loaded with the Modbus interface.

The table below lists the currently available controllers that can be ordered with the Modbus Gateway firmware.

Product Model	Description	Modbus Units
DSM-RTR-MOD30	BACnet Router w/ Modbus Gateway	30
DSM-T0B-MOD5-K1A	16 Button Room Controller w/ Modbus	5 [2]
DSM-T0B-MOD15-K1A	16 Button Room Controller w/ Modbus	15 [2]
DSM-T0B-MOD30-K1A	16 Button Room Controller w/ Modbus	30
DSM-PWR	Room Controller w/ Modbus Power Meter Interface	1

Notes: 1. This column lists the current number of Modbus slave devices supported by the associated controller.
2. Gateways not already maxed out to 30 can be flash loaded in the field to support additional Modbus units, up to 30.

Modbus Controllers

Almost any Delta I/O controller installed to control some mechanical equipment can be loaded with the Modbus interface to pick up a few Modbus devices or units (i.e., variable speed drives) *as part of the same mechanical system*.

However, in contrast to the Gateway products, these controllers cannot be ordered with Modbus preloaded on them. Rather, order whatever DAC or DSC product you will be using (with the appropriate I/O for the mechanical system), and then load the Modbus interface in the field for the number of Modbus units (or devices) you will be connecting to, up to a maximum of 5 Modbus units per controller. Please note below the two criteria needed in order to load Modbus.

Flash Loading Modbus Units into a Device

Flash Loader has been modified to use "credits" which are programmed onto a new Flash Loader hardware key in Delta production. The number of credits is based on what is ordered, and decrements when an upgrade is "chargeable." As a result, the one same hardware key is now used for all upgrades starting with **V3.30 Flash Loader Release 3**, for the following product upgrades:

- DCU firmware
- DAC/DSC/DSM/DLC/ASM firmware
- Modbus Gateway Interface and units
- BACstats – FUTURE (when a new flash loadable model is released)

Modbus can be loaded into any Delta product that meets the following criteria.

1. The controller must currently be loaded with at least V3.30 Release 2 firmware or higher; and

2. Particularly for DAC models, the controller hardware must be one of the more recent revisions that have the larger memory (shipped in a box with a green dot on it, and includes a green Product Notice) – where flash memory has been increased to to 1 MB and the SRAM has been doubled to 256K.

Note: You can also determine whether a product has more memory when it is powered up and communicating by looking at its DEV object using ORCAview. Any product that includes the underscore as part of its Model Name on the Product tab (i.e., DAC_322) has the increased memory as required. You can even create a custom report in ORCAview to list this information – and one has already been provided on GEORGE for download.

The V3.30 Release 3 Flash Loader is rather smart and can figure out what it can and cannot do with the controller you've connected to, and whether the controller is one that Modbus can be loaded into, and indicates how many Modbus units have already been loaded and and whether you can load any more.

Loading Modbus into a controller will use (or decrement) 10 credits off the Flash Loader key for each Modbus unit. Other product upgrades require a different number of credits. And once loaded into a Delta product, the credits are unrecoverable (you cannot reverse the process).

Refer to Flash Loader documentation or the Help file for further information, including how many credits are used to upgrade all other devices.

Flash Loader Key & Credits

As described, the same Flash Loader key can be used to upgrade V2 DCUs or DACs to V3, and can also be used to load the Modbus Gateway interface.

So, besides obtaining the right controller, you will need to order a Flash Loader key and perhaps additional credits, especially if you intend on upgrading a large number of controllers.

Model Number	Description
DFF099-KEY	FLASH LOADER FIRMWARE KEY **c/w 50 credits** already loaded. At 10 credits per Modbus unit, 50 credits will allow you to load 5 Modbus units into the same or different controllers.
DFF099-CDT	FLASH LOADER CREDITS - Additional credits you wish to add to the original 50 on the same key. Note that the quantity for this product represents blocks of 50. You must include this second part number in the same order if you want more than the default 50 credits on the key.

Connecting to the Modbus Device

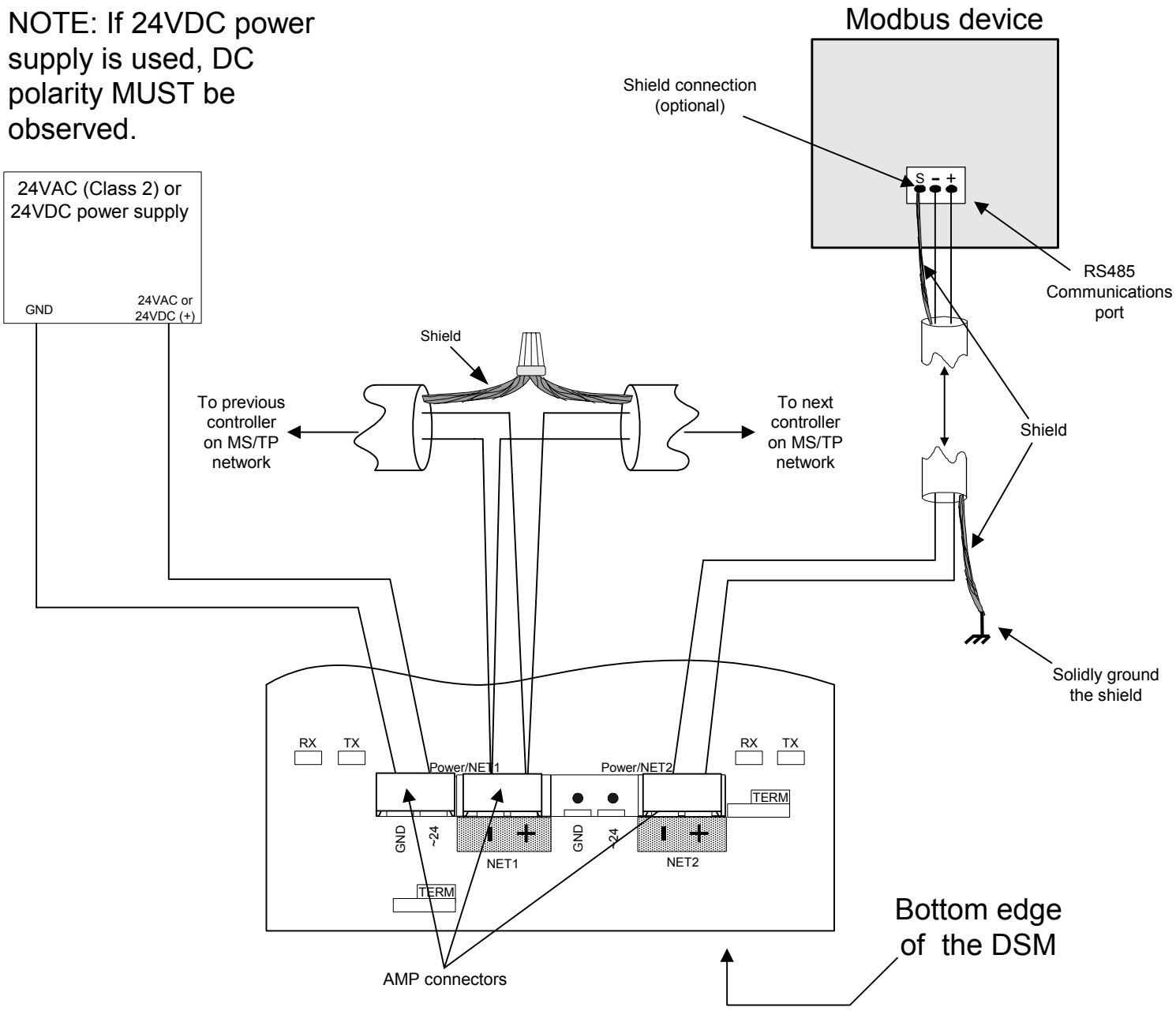
The Modbus Gateway controllers can connect to Modbus slave device(s) on either of their RS-485 ports (NET1 or NET2) or RS-232 (PTP) port. The table below shows the available configuration options for the NET and PTP ports on the Modbus Gateway controllers:

	NET1	NET2	PTP
Ethernet DSC or DSM-RTR	LINKnet or Modbus	BACnet MS/TP (Subnet Level) or Modbus	OWS Direct-Connect, Modem, or Modbus
Non-Ethernet DSC or DSM-T0B or DSM-PWR	BACnet MS/TP (System Level)	LINKnet or Modbus	OWS Direct-Connect, Modem, or Modbus
DAC+	BACnet MS/TP (Subnet Level)	LINKnet or Modbus	N/A

***Note:** When connecting to a Modbus device using the RS-232 (PTP) port only a single Modbus slave device is supported. To communicate with multiple devices requires an RS-485 (NET) connection to the Modbus slave devices.

The diagram on the following page shows how to wire a single Modbus slave device to a DSM-T0B-MODxx controller using the NET2 port. For more detailed wiring information refer to the Delta Controls Wiring Guidelines.

NOTE: If 24VDC power supply is used, DC polarity MUST be observed.



Note: *Use a shielded, twisted pair wire for communications. Never directly ground more than one point on the shield. Doing so can induce large currents, and result in communication problems.*

Configuring the Modbus Gateway

Two objects are required in order to make the controller communicate with the Modbus device(s). They are the Gateway (GW) and Gateway Translation (GWT) objects.

This section gives a brief explanation of these objects and shows how configure them for use.

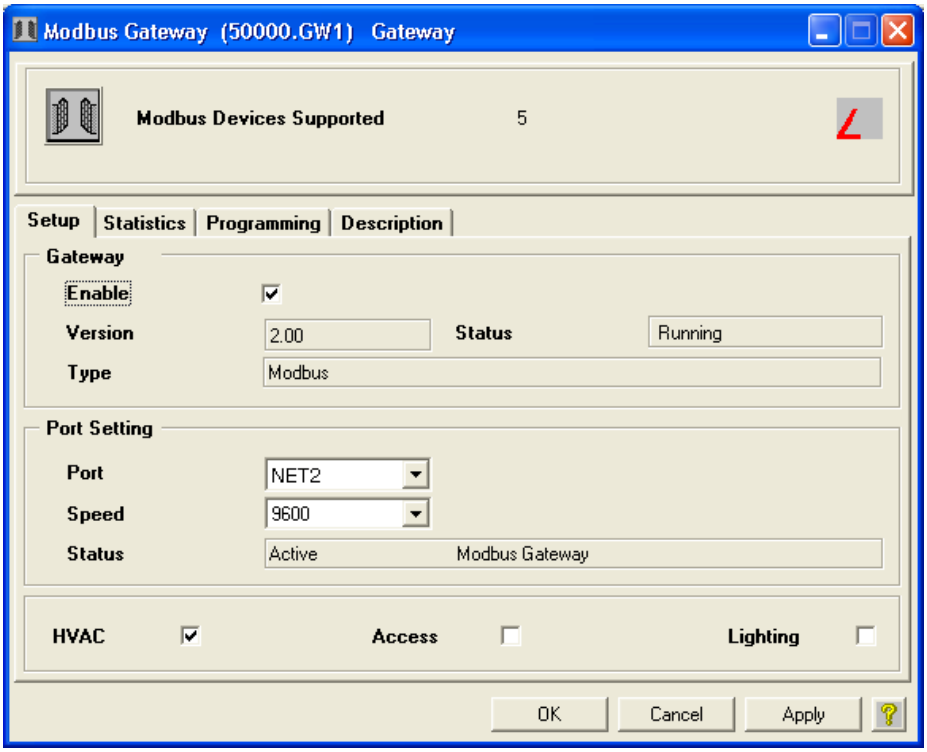
It is assumed the user already has a basic familiarity with the Modbus protocol, as well as the GW and GWT objects. For detailed information on the Delta Controls implementation of the Modbus protocol, as well as the GW and GWT objects, refer to the associated sections in the appendices of this document.

Communications Settings

The Gateway (GW) object is used to setup the interface between the Gateway controller and the Modbus device(s). You cannot create the GW object. It is automatically created in the controller whenever the Modbus Gateway firmware is present.

To setup the GW object, follow these steps:

1. Open the GW object and go to the setup tab.



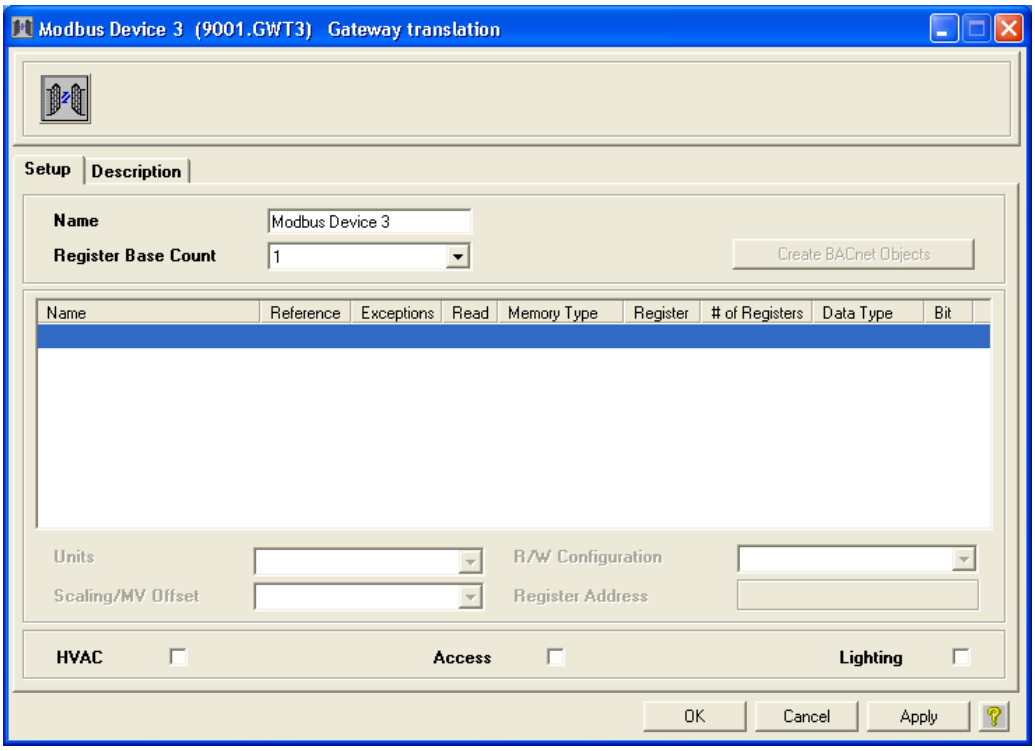
2. Ensure the Enable checkbox is checked and select from the Port drop down menu which port the Modbus device(s) are physically connected to. Using the Speed drop down menu select the communications baud rate of the Modbus device(s). (Refer to documentation that came with the Modbus device(s) for information on the valid connection speeds for that device.)

3. Press Apply and reset the controller.

Linking Modbus Device Registers to BACnet Objects

The GWT object is used to transfer values between Modbus registers and the associated BACnet objects. Refer to documentation that came with the Modbus device(s) for information on the registers. For each Modbus device, a GWT object must be created with an instance number equal to the network address of the Modbus device.

1. Create a GWT object with an instance number equal to the network address of the Modbus device to be associated with it.



2. From the documentation that came with the Modbus slave device, determine if it uses a 0 or 1-based register addressing convention and set the Register Base Count Field to match. (While Modbus specifies a 1-based register addressing convention, some implementations use a 0-based register addressing convention in their documentation of the register mappings. The Register Base Count field is used to configure the GWT to be compatible with the convention used by that particular device.)

3. In the dataview of the GWT object, create an entry for each value from the Modbus device to be linked to a BACnet object in the Gateway. Start entering data by double clicking in the column you wish to edit.

4. Below the dataview there are also several dropdown boxes that contain more configuration settings for the current selected entry. After filling out the fields in the dataview for a single entry, you must click on Apply before you can enter data in these remaining fields. After you finish setting these additional fields press Apply again to save the changes before moving to the next entry.

5. Press the Create BACnet Objects button and press Apply. This will automatically create all the required BACnet objects based on the object references and names you entered in the steps above.

Note:
1) *If an object reference is already used in the database, the name of the object will not be overwritten by the GWT object, however a link will still be created to the Modbus register. If this object was already created because it is being used for some other purpose in the database, there will be a conflict since there would now be two competing control sources for the object.*
2) *Changes to the name or units properties for a given entry in the GWT after the associated BACnet object (AV, BV or MV) has been created have no effect. Once the object has been created, any changes to these properties should be done directly in the object itself.*

GWT Configuration Examples

This section provides several examples on how to configure the GWT to read from and/or write to the different Memory Types using various methods.

Note:
1) *The following examples all assume a Register base count setting of 1. If the register base count is set to 0, subtract 1 from the value to be entered in the Register field for each example.*
2) *If you wish to both read and write to a single value in a Modbus device, you must create two separate entries in the table one for reading the value and another for writing to the value. In addition, these two entries must not be referencing the same BACnet object.*

Reading Input Status Registers

Input Status Registers are single bit read only registers that store a binary (On or Off) value. The following table shows the GWT settings required to Read the value from Input status register 10001 into BV1 in the Gateway Controller.

Example:

Read Input Status Register 10001 to BV1

Reference	R/W Config.	Memory Type	Register	# of Registers	Data Type	Bit
BV1	Read	Input Status	1	1	Bit Value	N/A

Reading/Writing Coil Registers

Coil Registers are similar to Input Status Registers, except they are writeable. The example below shows the GWT settings to write to Coil Register 00001.

Example:

Write BV1 value to Coil Register 00001

Reference	R/W Config.	Memory Type	Register	# of Registers	Data Type	Bit
BV1	Write	Coil	1	1	Bit Value	N/A

Reading Input Registers

Input Registers are read only 16-bit registers that can hold analog values in several different data formats. The data format will generally determine what type of object the value should be mapped to and the number of registers that should be read. The table below shows the valid settings for the object type, number of registers to be read and bit fields based on the data type selected.

Object Type	# of Registers	Data Type	Bit
BV	1	Bit Value	0-16
AV	1	Integer	N/A
	2	Long	N/A
	1	Bit Value	0-16
	2	Double Integer	N/A
	2	Float	N/A
	2	FloatSwapped	N/A
	2	FloatReversed	N/A
MV	1	Integer	N/A

(For a definition of the different Data Types see the Data Types section of Appendix C)

Examples:

Read a Floating Point Value from Input Register 30002 to AV1

Reference	R/W Config.	Memory Type	Register	# of Registers	Data Type	Bit
AV1	Read	Input Register	2	2	Float	N/A

Read Bit 2 from Input Register 30002 to BV1

Reference	R/W Config.	Memory Type	Register	# of Registers	Data Type	Bit
BV1	Read	Input Register	2	1	Bit Value	2

Reading/Writing Holding Registers

Holding Registers are the same as Input Registers with two exceptions:

1. Holding Registers can be written to as well as read from.

2. You cannot use the Bit Value data type when writing to a Holding register. If you want to write to a single bit in a holding register, you should read out the register value as an integer into an AV, then use GCL bit manipulation to set/clear the required bit and write this adjusted value back into the Modbus device.

Examples:

Read an Integer Value from Holding Register 40001 to AV1

Reference	R/W Config.	Memory Type	Register	# of Registers	Data Type	Bit
AV1	Read	Holding Register	1	1	Integer	N/A

Write the Value from AV3 as a Long to Holding Register 40002

Reference	R/W Config.	Memory Type	Register	# of Registers	Data Type	Bit
AV3	Write	Holding Register	2	2	Long	N/A

Memory Limits & Performance Issues

Each entry (register) in a GWT object occupies approximately 100 bytes of Static RAM (database space). Given that a DSM-RTR has 319 kB of Static Ram available, this puts an ultimate cap of about 3,200 entries across all created GWT objects. However, this would not allow for any trending, alarming, programming, etc. because the memory would be totally full.

Since a DSM-RTR-MOD30 can communicate with up to 30 Slave Devices, this would nominally allow each slave to have about 100 entries (registers)

[100 entries * 30 GWT objects = 3000 total entries].

However, with this many entries, it will take substantial time for the DSM-RTR-MOD30 to read all of the values from all the Modbus slaves.

This has been estimated to take 13 minutes. So, clearly, control of these slave devices will be impossible, since new command signals could only be sent once every 13 minutes! However, if these 30 slaves were power meters, then this might be perfectly acceptable.

If however, VFDs were to be controlled via Modbus (start/stop and speed), then a cautious approach is advised. 30 VFDs that each have 10 registers (total of 300 registers) would require about 75 seconds to update. Pressure control (either liquid or air) will be nearly impossible at this update interval. Even with a minimum of 3 registers per VFD (start/stop, speed control, speed status) (total of 90 registers), it would take about 22 seconds to update all the VFDs. This might be acceptable for air pressure control, but not for liquid pressure control.

Unlike previous firmware versions, it is now possible to write to however many Modbus registers is required. There is no arbitrary limit as there had been in the past.

The GWT object is used to transfer values between Modbus registers and the associated BACnet objects. Refer to documentation that came with the Modbus device(s) for information on the registers. For each Modbus device, a GWT object must be created with an instance number equal to the network address of the Modbus device.

Appendices

Appendix A: Troubleshooting

There are several features in this version of the Delta Modbus Gateway that can be used to help when troubleshooting the data exchange between the Modbus Gateway and it's associated devices.

Gateway Object Statistics Tab

The statistics tab of the Gateway object has information that is very useful when troubleshooting. See the GW Object Reference Statistics Tab Section in Appendix B for a description of the information contained in these fields.

Fault Flag Linked to a Modbus Register

When set, the fault flag of a BACnet object, which is linked to a Modbus register, indicates a problem with reading from or writing to that register value. There are three possible causes for this:

1) There is no response from the associated Modbus device.

   This typically means the device is offline. Ensure the device is powered up and verify the communications wiring between the devices. (If connecting using one of the NET ports check that both the Tx and Rx lights are blinking.) Next verify that you have the correct port and speed settings in the GW object.

2) There was an exception code returned from the Modbus device.

   Check the Exceptions field in the dataview entry of the associated GWT object. If it is non-zero then an exception was returned from the slave device. This means the slave device had a problem executing the requested query from the Gateway controller. (Refer to the Exception Responses section of Appendix C for more information.)

3) The associated entry in the GWT for this object contains an invalid combination of settings. (For example a data type of float cannot be mapped to a BV.)

   Refer to the tables in the GWT Configuration Examples section for details on the valid settings based on data type.

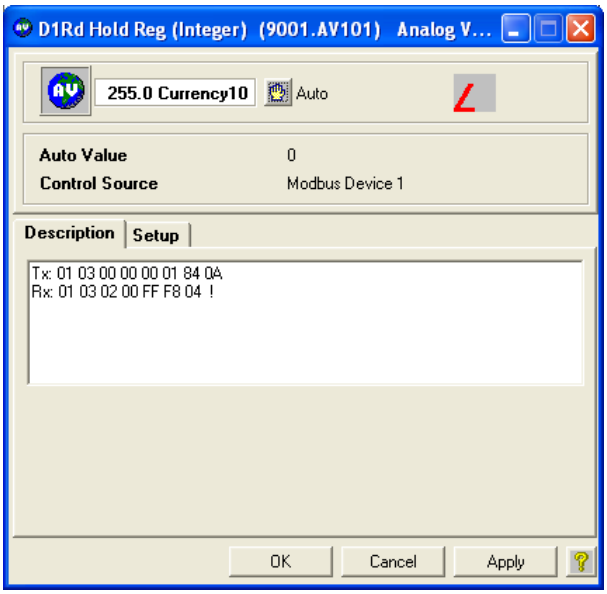
Exception Code Responses

Exception codes may be returned when there is a problem with the request that was received by a Modbus slave device. The code returned in the exception response gives diagnostic information on the reason why the request failed to process normally.

(For more information and a list of the exception codes defined by the Modbus specification see the Exception Responses section in Appendix C)

Tx & Rx Packet Snooping

Setting the units for the associated object to Currency 10 enables a special 'packet sniffing' mode. When this mode is enabled the description tab of the associated BACnet object for a given GWT entry will display the Tx (transmit) packet sent out to the Modbus device on the last scan for that entry. It will also display any Rx (receive) packet response sent back from the Modbus device for that entry. The following screen capture shows the Rx and Tx packets in the description tab for reading an integer value from Holding Register 40001 of a Modbus device with an address of 1.



(See Appendix C: Modbus Protocol Implementation for details on the packet structure for Modbus communications.

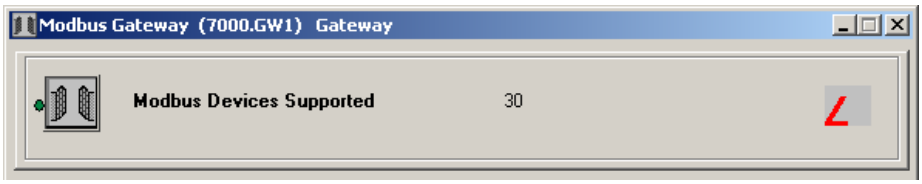
Note:
1) *Any text already in the description tab when packet sniffing mode is enabled will be overwritten.*
2) *For AV object's you must change the units directly in the associated AV object to go to packet sniffing mode for that entry. For BVs and MVs, which do not have a units field, change the units property for the associated entry in the GWT object.*

Appendix B: GW and GWT Object Reference

The following sections give detailed information on all the application specific fields in the Gateway (GW) and Gateway Translation (GWT) Objects as they appear when opened in a Modbus Gateway Controller.

Gateway Object (GW)

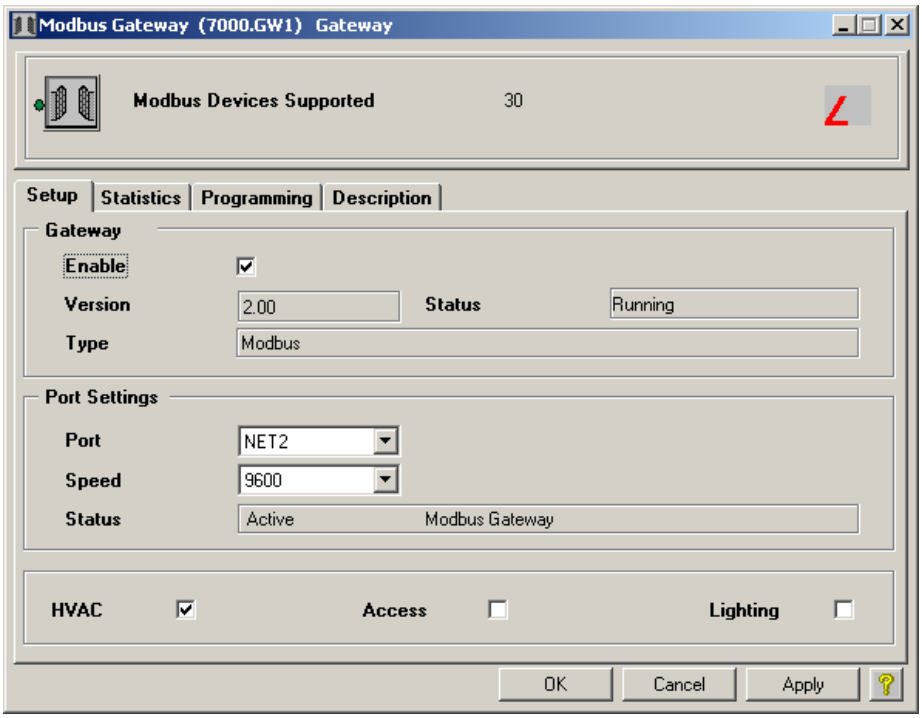
Header



Modbus Devices Supported

This field displays the maximum number of Modbus slaves the controller is able to communicate with.

Setup



Enable

This checkbox enables or disables the gateway.

Version

This field displays the version number of the *gateway firmware* loaded into the Delta Controller, which is different from the main firmware version number listed in the Device object.

Type

This field displays the type of gateway that is loaded into the Delta Controller. In V3.30A firmware there are only two gateways currently supported:

- Modbus
- Delta Wireless Sensors

Status

This field displays the status of the gateway. When the gateway is enabled, this field displays Running. When the gateway is disabled, this field displays Stopped.

Port

This field is used to select the physical port which is to be connected to the Modbus slave device(s).

There are three port options available:

- NET1 (RS-485 port 1)
- NET2 (RS-485 port 2)
- PTP (Serial/PTP port)*

**** Note:***
1) *Only DSC and DSM controllers come with a serial port.*
2) *When connecting via PTP only a single Modbus slave device is supported.*

Speed

This field is used to select the baud rate for communicating with the Modbus slave devices(s). The baud rate must be set to the same speed on both devices in order that they communicate properly. The supported speeds are 2400, 4800, 9600, 19200, and 38400 baud.

Status

This field displays the current status of the selected port. When the port is being used by the gateway, this field displays Active. When the gateway is disabled, this field displays Inactive. If the selected port is in use by something other than the gateway (i.e. NET object), this field displays "In Use By" and lists the name of the object using the port.

HVAC, Access, and Lighting Checkboxes

These checkboxes indicate which application(s) the object is part of. Any combination of the HVAC, Access Control, and Lighting applications can be selected. Leaving all checkboxes blank is the same as checking them all.

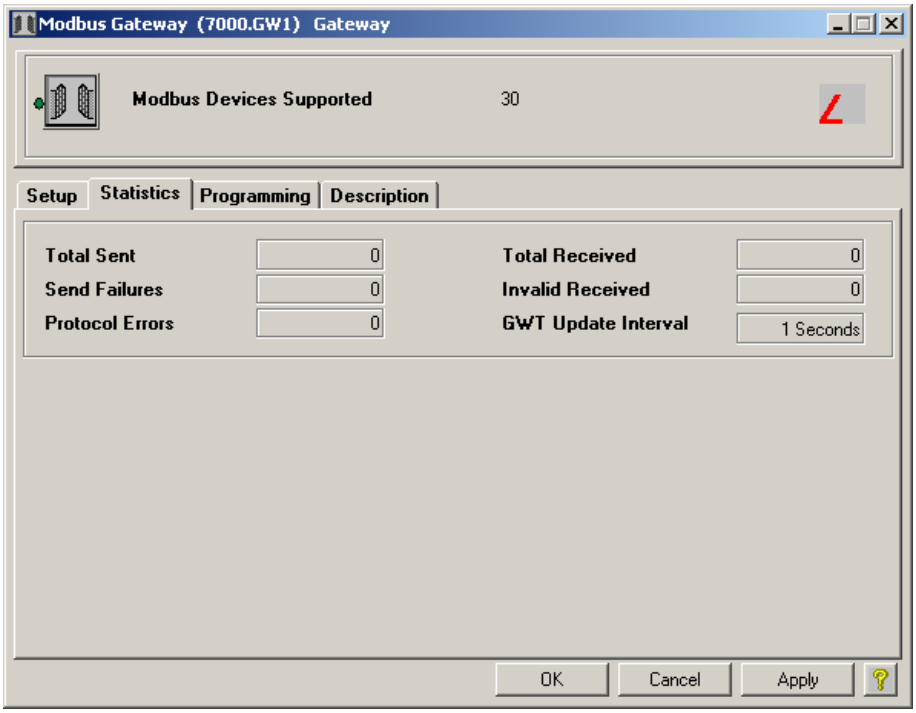
As a part of OWS security, these checkboxes allow an operator to see only the objects for a particular application(s). For example, a site may have separate operators for the HVAC and Access applications. It is possible for each operator to only see the objects related to their application.

Refer to Chapter 7 - Security in the ORCAview Technical Reference Manual for more information on how to set up and administer Object Application Restrictions

When a checkbox is checked, an indicator icon appears at the top of the dialog. The three application domains are:

Application	Icon
HVAC (Red)	
Access (Blue)	
Lighting (Green)	

Statistics



Total Sent

This field displays the total number of Modbus packets sent from the Delta Controller since the last controller reset. The value of this number is not very important but it should be continuously incrementing. If the Total Sent is not incrementing, then it means that no packets are being sent.

Total Received

This field displays the total number of packets received from the Modbus slave(s) since the last controller reset. The value of this field should be incrementing along with the Total Sent field. If the Total Received is not incrementing along with the Total Sent field, then it means that the Delta Controller is sending requests but the Modbus slaves are not responding.

Send Failures

This field displays the total number of sent packets that failed to receive a response since the last controller reset.

Invalid Received

This field displays the total number of invalid packets received since the last controller reset. Invalid packets are those that contain an incorrect cyclic redundancy code (CRC).

A CRC is a number derived from, and transmitted with, a block of data in order to detect corruption. By recalculating the CRC and comparing it to the value originally transmitted, the receiver can detect some types of transmission errors.

Protocol Errors

This field displays the total number of exception responses returned since the last controller reset.

GWT Update Interval

This is the time in seconds that it took on the last iteration to scan all the GWT objects and update the associated BACnet objects. The update rate varies depending on the number of Modbus slave devices, and how many values are mapped from each device.

Programming

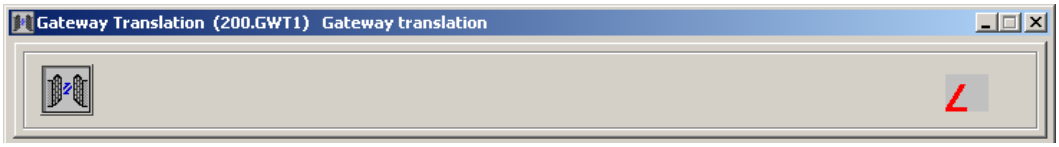
The Programming tab is not used for the Modbus Gateway but only with the Delta Wireless Gateway.

Description

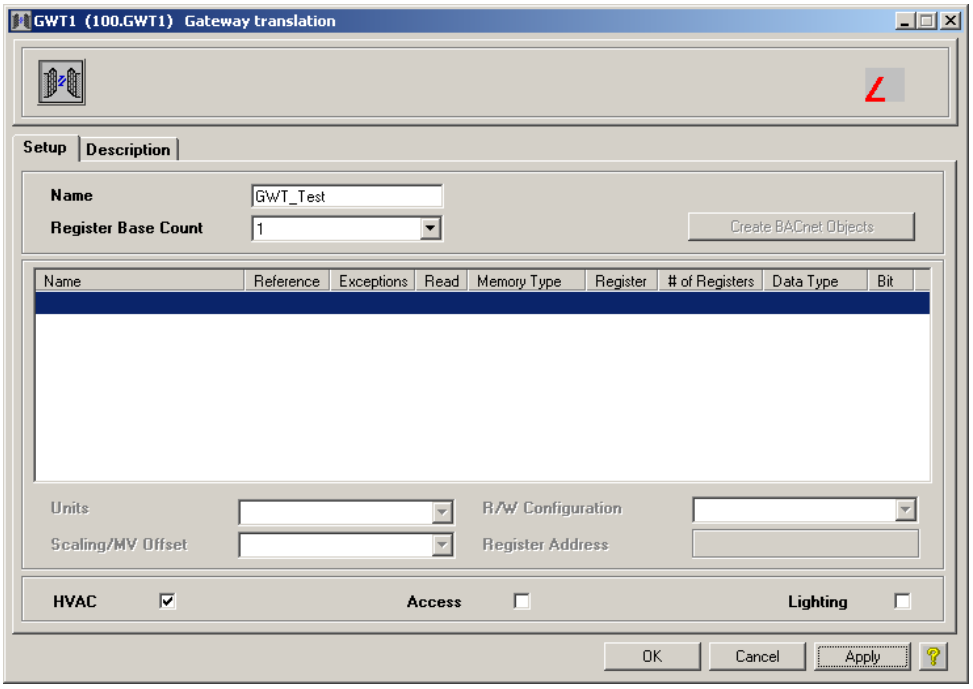
The description tab may contain a string of characters of up to 2000 characters in length. The character string is limited to any printable character.

Gateway Translation Object (GWT)

Header



Setup (Common Area)



The common area of the Setup tab has two fields and a button located above the dataview area.

Name

The name of the Gateway Translation object is displayed in this field. This name must be unique among all the descriptors located in the same controller.

Register Base Count

While Modbus specifies a 1-based register addressing convention, some implementations use a 0-based register addressing convention in their documentation of the register mappings. The Register Base Count field allows the GWT object to be compatible with either convention.

For example, the first register in the Holding Registers memory block (also refered to as 4x references) may be documented any of the following ways:

- Holding Register 1 (1-Based Decimal Register Count)
- Holding Register 0 (0-Based Decimal Register Count)
- Holding Register 0000H (0-Based Hexadecimal Register Count)
- Register 40001*

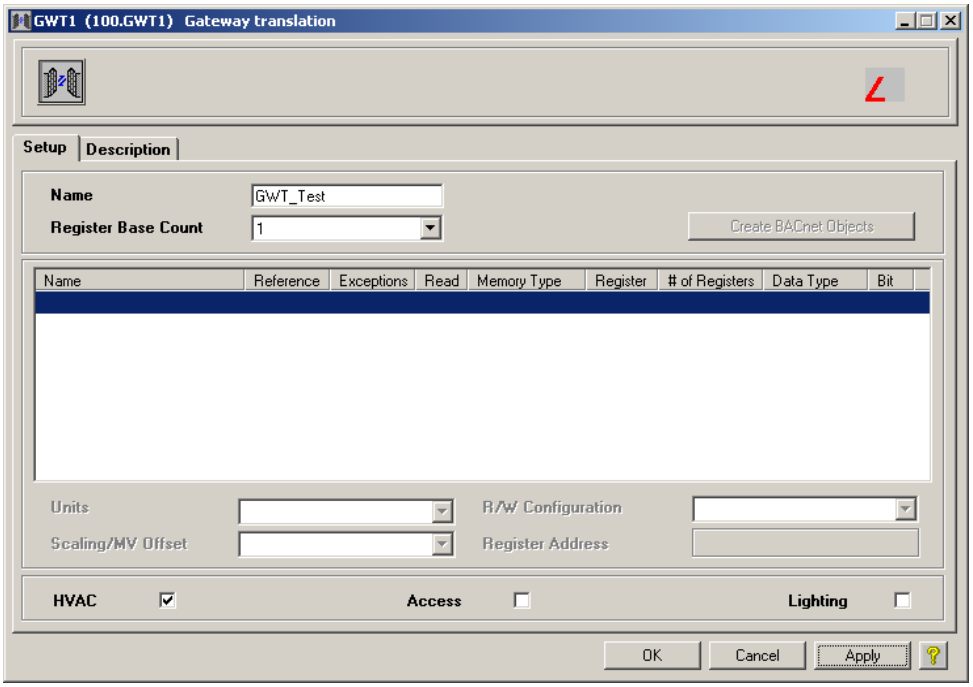
** **Note**: The leading digit in the last example is used to represent the memory type (in this case a Holding register) while the remaining digits indicate the register count in a 1-based decimal format.*

Refer to the Memory Types section of Appendix C and the documentation that came with the Modbus device(s) to determine which setting to use.

Create BACnet Objects Pushbutton

This button is used to automatically create the objects that are listed in the Reference column of the dataview if they do not already exist. This button has to be depressed and the Apply button pressed in order to create the objects.

Setup (Dataview)



The Data View on the Setup tab lists all of the translated information from the Modbus device.

Clicking on one of the entries will display additional information below the Data View.

Name

The Name of the BACnet object that is linked to the Modbus register for this entry.

Reference

The object reference of the associated BACnet object. Only AV, BV, and MV object references are supported.

Exceptions

A read only field that shows any exception codes returned from the Modbus device for that entry. Exception codes are error codes returned from a Modbus slave device in response to a request from the Master device.

Refer to Exception Responses in Appendix C for more information.

Read

When the associated checkbox is checked this indicates that the value is being read from the Modbus device into the associated BACnet object. Unchecked indicates the value is being written to the Modbus device. This field is read-only in the dataview. To change this value you must highlight the desired entry and use the R/W Configuration drop-down box located below the dataview.

Memory Type

Memory Type determines where the data is stored in the memory of the Modbus device. Each Memory Type reads or writes to a certain range of memory in the Modbus device. The different Memory Types and corresponding register reference are listed below.

- *Coil Register* – 0x References
- *Input Status Register* – 1x References
- *Input Register* – 3x References
- *Holding Register* – 4x References

Refer to the Memory Types in Appendix C for more information.

Register

Specifies the starting register for the data value in the Modbus device to be referenced. Registers are specified without the Memory Type reference (i.e. 0x, 1x, 3x, 4x). For example, for a Memory Type register reference of 40001, enter a value of 1 assuming a 1-based Register Count.

# of Registers

Specifies the number of registers that the data value in the Modbus device spans. The value that is entered in this field is determined by the data format selected in the Data Type field.

Data Type

Specifies the data format used to read/write the data value stored in the Modbus device. The different Data Types are:

- *Bit Value* – 1-bit Boolean (ON/OFF) (occupies 1 register)
- *Integer* – 16-bit signed Integer (occupies 1 register)
- *Long* – 32-bit signed Integer (occupies 2 register)
- *Double Integer* – 32-bit Unsigned Integer (occupies 2 registers)
- *Float* – IEEE 32-bit floating point Big Endian (occupies 2 registers)
- *Float (Reversed)* – IEEE 32-bit floating point Little Endian (occupies 2 registers)
- *Float (Swapped)* – IEEE 32-bit floating point Little Endian with byte-swapped (occupies 2 registers)

Refer to the Data Types in Appendix C for more information.

Bit

Specifies a particular bit to extract when reading from a 16-bit Holding or Input register. This can be a value from 0-16 were 1 is the least significant bit (LSB) and 16 is the most significant bit (MSB) in the selected register. 0 is a special case in which the entire 16-bit register is evaluated as either zero or non-zero. This field is only used when mapping a value from a Holding or Input register to a BV with a data type of Bit Value.

Setup (Dataview Fields)

The following four fields are not actually in the dataview window itself. They are located directly below the dataview in the GWT dialog.

Units

The Units drop-down box is used to select the units that the associated BACnet object will be created with when the Create BACnet Objects pushbutton is used. This field only applies when mapping to an AV object. MV or BV objects do not have a units property.

R/W Configuration

This dropdown box is used to select whether you want to read from or write to the selected data value in the Modbus device.

Note: *If you wish to both read and write to a single value in a Modbus device, you must create two seperate entries in the table, one for reading the value and another for writing to the value. In addition, these two entries must not be referencing the same BACnet object.*

Scaling/MV Offset

This field is used when mapping a Modbus value to an AV or MV object.

When mapping to an AV object, it can be used to rescale a value from a Modbus device by doing a left or right shift of up to 4 decimal places. For example, if the Modbus device returned a value of 600 which represents a value of 60.0Hz, selecting Div10 from this dropdown would convert the value from 600 to 60.0 before placing it into the associated AV object.

MV objects take an integer value and convert it to a text value based on an enumeration list in an associated MIC object. The MIC enumeration list is 1 based while some Modbus device enumerations are 0 based. In order to map a 0 based enumeration value from a Modbus device to an MV object, select Div10 (or +1 MV Offset) from the dropdown list.

Register Address

This is a display only field that displays the full decimal and Hexadecimal Address for the starting register of the data in the Modbus device for the currently selected dataview entry.

HVAC, Access, and Lighting Checkboxes

These checkboxes indicate which application(s) the object is part of. Any combination of the HVAC, Access Control, and Lighting applications can be selected. Leaving all checkboxes blank is the same as checking them all.

As a part of OWS security, these checkboxes allow an operator to see only the objects for a particular application(s). For example, a site may have separate operators for the HVAC and Access applications. It is possible for each operator to only see the objects related to their application.

Refer to Chapter 7 - Security in the ORCAview Technical Reference Manual for more information on how to set up and administer Object Application Restrictions

When a checkbox is checked, an indicator icon appears at the top of the dialog. The three application domains are:

Application	Icon
HVAC (Red)	
Access (Blue)	
Lighting (Green)	

Description

The description tab may contain a string of characters of up to 2000 characters in length. The character string is limited to any printable character.

Appendix C: Modbus Protocol Implementation

The following sections give general information about the Modbus protocol and specifics on what portions of the protocol Delta has implemented in its Modbus Gateway controllers.

Protocol: Modbus RTU (The Modbus ASCII protocol is not supported by this Gateway.)

Byte format: 8N1 (8 data bits, no parity and 1 stop bit), with CRC error checking. Modbus devices that require a Byte format with even or odd parity and 2 stop bits are not supported.

Connection Speeds: Connection speeds of 2400, 4800, 9600, 19200 and 38400 baud are supported.

Functions: The following Modbus functions are supported. (Refer to sections following this for more detailed information on Modbus functions, memory types and packet structure.)

Code (Decimal)	Name
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input Registers
05	Force Single Coil
06	Preset Single Register
16	Preset Multiple Registers

Memory Types

Modbus data values are broken into 4 different memory types each of which represent a block of memory in the Modbus device. Each of these blocks of memory is broken up into registers. Depending on the memory type and data format, a single data value will span 1 or more registers. To access a particular register, both the register number and memory type must be specified. Together these represent the register address. There are several different conventions in which register addresses are documented:

For example, the first register in the Holding Registers memory block (also referred to as 4x references) may be documented any of the following ways:

- Holding Register 1 (1-Based Register Count)
- Holding Register 0 (0-Based Register Count)
- Holding Register 0000H (0-Based Hexadecimal Register Count)
- Register 40001

A brief description of each memory type is given below:

Coil Registers (0x References) – Each coil register is a 1-bit piece of data that typically represents the Boolean (ON/OFF) state of an output or internal variable. These registers can be written to or read from.

Input Status Registers (1x References) – Each Input Status register is a 1-bit piece of data that typically represents the Boolean (ON/OFF) status of an input or internal variable. These registers are read-only.

Input Registers (3x References) – Each Input register is a 16-bit piece of data one or more of which typically represent the analog status of an input or internal variable. There are several different formats in which the data may be stored. (These different data formats and the number of registers each takes is covered in the Data Types section of this appendix.) These registers are read-only.

Holding Registers (4x References) – Each Holding register is a 16-bit piece of data one or more of which typically represent the analog state of an output or internal variable. There are several different formats in which the data may be stored. (These different data formats and the number of registers each takes is covered in the Data Types section of this appendix). These registers can be written to or read from.

Data Types

The data values stored in Modbus registers can be in several formats. You need to know in what format the information is stored in order to be able to correctly configure the GWT object to read and/or write that value. This section details the different data formats supported by the Delta Modbus Gateway.

Bit Value – Bit values are stored in a single 1-bit register. They are used to store simple binary values. (1 = ON, 0 = Off)

Integer – This data type uses a 16-bit signed integer format to store a whole number value from –32768 to 32767. Integer values are stored in a single 16-bit register.

Double Integer – The double integer data type spans two 16-bit registers and can store any positive whole number value from 0 to 4,294,967,295

Long – Longs are 32-bit signed Integers. In this data type the most significant bit (MSB) is used to indicate if the value is positive or negative. The remaining 31 bits are the binary equivalent of the decimal value. For example, 3 and –3 expressed in Binary Long format would be:

Decimal Value	Binary 32-bit signed Integer Format
3	00000000000000000000000000000011
-3	10000000000000000000000000000011

A Long value spans two 16-bit registers and can store any whole number value from -2,147,483,648 to +2,147,483,647.

Note: *Delta uses a 32-bit (single precision) floating-point format to store BACnet AV values. This limits the stored value to 6 significant figures plus an exponent value. This means that when mapping Double Integer and Long values from a Modbus device to AV's you are actually limited to integer values between –340,282 and +340,282 before round off errors due to trailing significant figures being dropped occurs.*

32-Bit (Single Precision) Floating Point numbers – Floating-Point values are used to represent real numbers (numbers with a fractional component). IEEE specifies a standard format for 32-bit floating-point values. There are 4 variations in which IEEE Floating-Point values may be stored and/or transmitted in a data packet. The Delta Modbus Gateway supports 3 of these formats. The table below lists the 3 formats that Delta supports and which data type should be selected in the GWT to reference a floating-point value using this format.

Data Type	Data Format
Float	Floating Point Big Endian
Float (Reversed)	Floating Point Little Endian
Float (Swapped)	Floating Point Little Endian with byte-swapped

Single Precision (32-bit) floating-point values span two 16-bit registers and can store a real number value from 3.403×10^{-38} to 3.403×10^{38}.

IEEE Floating Point Data Format Details

This section gives a detailed description of the IEEE 32-bit floating-point data format.

The IEEE floating-point data format is similar to binary scientific notation data format. Scientific notation expresses real numbers in the following format:

$$\text{Mantissa } \mathbf{x} \text{ Number Base}^{\text{Exponent}}$$

For example, lets convert 146.5 to scientific notation using a number base of 10 (decimal):

1. Start by calculating the Mantissa. To do this, take the number to be converted, in this case 146.5, and shift the decimal point until there is only one digit to the left of it. This is called Normalizing the Mantissa. This gives a value of 1.465
2. Next calculate the exponent by counting how many decimal places you shifted to normalize the mantissa. If the decimal was shifted to the left the exponent is positive and if shifted to the right it is negative. In this case the exponent is +2.
3. By using the formula above, the scientific notation equivalent of 146.5 is:

$$\text{Mantissa } \mathbf{x} \text{ Number Base}^{\text{Exponent}} = \mathbf{1.465 \times 10^2}$$

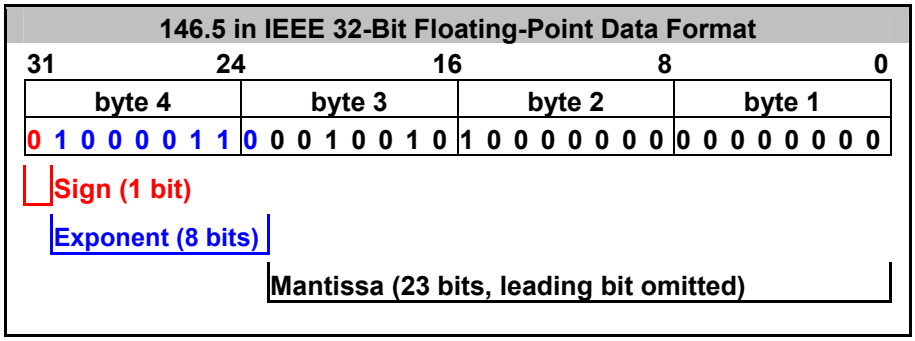
Binary scientific notation is done exactly the same way only using binary (base 2) numbering. For example, the decimal number 146.5 in binary is 10010010.1. Normalizing the Mantissa gives a value of 1.00100101. The decimal was shifted 7 places to the left giving an exponent value of +7. Therefore, 146.5 in binary scientific notation is:

$$\text{Mantissa } \mathbf{x} \text{ Number Base}^{\text{Exponent}} = 1.00100101 \times 2^7$$

IEEE 32-bit Floating-Point notation is similar to binary scientific notation with a few exceptions:

1. One bit is used to indicate the sign of the value. (Positive or Negative)
2. The exponent value has a +127 bias added to it.
3. In a base 2 normalized mantissa, the digit on the left of the decimal point will always be 1. In IEEE format only the fractional portion of the Mantissa is stored and it is understood there is always an implied 1. in front of it.

The following diagram shows how an IEEE floating-point value is stored in a 32-bit binary number using our example value of 146.5.



Sign – (bit 31) The sign is stored as a single bit Boolean value (0 = Positive, 1 = Negative)
Exponent – (bits 23-30) The exponent is stored as an 8-bit unsigned integer value.
Mantissa – (bits 0-22) The fractional part of the Mantissa stored as a 23-bit binary fraction.

As stated earlier there are 4 variations on the above data format. The difference between each of these variations is the order in which each byte of data (as numbered in the table above) is stuffed into a memory location and/or data packet. The table below gives the stuffing order for each of the 4 formats:

Data Format	Byte Order
Floating Point Big Endian	4,3,2,1
Floating Point Big Endian with byte-swapped*	3,4,1,2
Floating Point Little Endian	1,2,3,4
Floating Point Little Endian with byte-swapped	2,1,4,3

***Note:** Floating Point Big Endian with byte-swapped format is not supported in the Delta Modbus Gateway.

Functions

The Delta Modbus Gateway Controllers support functions 1-6 and 16 of the Modbus protocol. This section provides detailed information on each of these functions.

Note: *Register addresses in the data packets are always zero-based regardless of the Register Base Count convention being used in documentation.*

Function 01 Read Coil Status

This function is used to read the ON/OFF state of a coil register.

In a Read Coil Status Query message the starting register address and number of registers to be read are specified.

Note: *While this function allows reading of multiple register values in a single query, Delta only supports reading a single register per query message with this function.*

The following example shows the packets generated when requesting to read coil register 00011 from a Modbus slave with an address of 2:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	02
Function	01
Starting Address Hi Byte	00
Starting Address Lo Byte	0A
# of Points Hi Byte	00
# of Points Lo Byte	01
Error Check (CRC)	-

The normal response if the queried coil register was in the on state would be:

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	02
Function	01
Data Byte Count	01
Data	01
Error Check (CRC)	-

In the Data byte the information on the coil state is stored in the least significant bit (LSB). A value of 1 = On, 0 = Off, and the remaining bits in the data byte are padded with zeros.

Function 02 Read Input Status

This function is used to read the ON/OFF status of an Input Status Register.

In a Read Input Status Query message the starting register address and number of registers to be read are specified.

Note: *While this function allows reading of multiple register values in a single query, Delta only supports reading a single register per query message with this function.*

The following example shows the packets generated when requesting to read Input Status register 10197 from a Modbus slave with an address of 25:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	19
Function	02
Starting Address Hi Byte	00
Starting Address Lo Byte	C4
# of Points Hi Byte	00
# of Points Lo Byte	01
Error Check (CRC)	-

The normal response if the queried Input Status register was in the off state would be:

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	19
Function	02
Data Byte Count	01
Data	00
Error Check (CRC)	-

In the Data byte the information on the Input Status state is in the least significant bit (LSB). A value of 1 = On, 0 = Off, and the remaining bits in the data byte are padded with zeros.

Function 03 Read Holding Registers

This function is used to read the contents of 1 or more 16-bit Holding Registers.

In a Read Holding Register Query message the starting register address and number of registers to be read are specified.

Note: *While this function allows reading of multiple register values in a single query, Delta only supports reading of up to 2 registers in a single query (depending on the data type).*

The following example shows the packets generated when requesting to read Holding registers 40108-40109 from a Modbus slave with an address of 5:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	05
Function	03
Starting Address Hi Byte	00
Starting Address Lo Byte	6B
# of Points Hi Byte	00
# of Points Lo Byte	02
Error Check (CRC)	-

The normal response if holding registers 108 and 109 contained a 32-bit unsigned integer value of 169,824,461 would be:

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	05
Function	03
Data Byte Count	04
Data Byte Hi (Register 40108)	0A
Data Byte Lo (Register 40108)	1F
Data Byte Hi (Register 40109)	50
Data Byte Lo (Register 40109)	CD
Error Check (CRC)	-

Function 04 Read Input Registers

This function is used to read the contents of 1 or more 16-bit Input Registers.

In a Read Input Register Query message the starting register address and number of registers to be read are specified.

Note: *While this function allows reading of multiple register values in a single query, Delta only supports reading of up to 2 registers in a single query (depending on the data type).*

The following example shows the packets generated when requesting to read Input register 30108 from a Modbus slave with an address of 5:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	05
Function	04
Starting Address Hi Byte	00
Starting Address Lo Byte	6B
# of Points Hi Byte	00
# of Points Lo Byte	01
Error Check (CRC)	-

The normal response if Input register 108 contained a 16-bit unsigned integer value of 2591 would be:

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	05
Function	04
Data Byte Count	02
Data Byte Hi (Register 30108)	0A
Data Byte Lo (Register 30108)	1F
Error Check (CRC)	-

Function 05 Force Single Coil

This function is used to write the contents of a single coil register to either ON or OFF.

In a Force Single Coil Query message the starting register address and the value to write to the coil are specified. (FF 00 hex = ON; 00 00 hex = OFF)

The following example shows the packets generated when writing ON to Coil register 00172 to a Modbus slave with an address of 17:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	11
Function	05
Starting Address Hi Byte	00
Starting Address Lo Byte	AB
Force Data Hi Byte	FF
Force Data Lo Byte	00
Error Check (CRC)	-

The normal response is an echo of the query, returned after the register contents have been written to.

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	11
Function	05
Starting Address Hi Byte	00
Starting Address Lo Byte	AB
Force Data Hi Byte	FF
Force Data Lo Byte	00
Error Check (CRC)	-

Function 06 Preset Single Register

This function is used to write to the contents of a single holding (4x) register.

In a Preset Single Register Query message the starting register address and the data value to write to the register are specified.

Note: *Since this function can only write to a single register at a time, it is therefore only used when writing to Holding registers with a data type of Integer. The other data types, which span more than one register, write using function 16.*

The following example shows the packets generated when writing a value of 3 to holding register 40002 in a Modbus slave with an address of 17:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	11
Function	06
Starting Address Hi Byte	00
Starting Address Lo Byte	01
Force Data Hi Byte	00
Force Data Lo Byte	03
Error Check (CRC)	-

The normal response is an echo of the query, returned after the register contents have been written to.

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	11
Function	06
Starting Address Hi Byte	00
Starting Address Lo Byte	01
Force Data Hi Byte	00
Force Data Lo Byte	03
Error Check (CRC)	-

Function 16 (10 Hex) Preset Multiple Registers

This function is used to write to a series of holding (4x) registers.

In a Preset Multiple Registers Query message the starting register address, number of registers to be written to, the number of data bytes followed by the data value to write to the register are specified.

Note: *This function is used when writing values with a data type that spans more than one register to a holding register. (The Double Integer, Long and all the Float data types span two registers.)*

The following example shows the packets generated when writing a value of 3 in a Signed 32-bit data format to holding registers 40002 and 40003 in a Modbus slave with an address of 17:

Tx Packet (Query)

Field Name	Example (Hex)
Slave Address	11
Function	10
Starting Address Hi Byte	00
Starting Address Lo Byte	01
# of Registers Hi Byte	00
# of Registers Lo Byte	02
Byte Count	04
Data Hi	00
Data Lo	00
Data Hi	00
Data Lo	03
Error Check (CRC)	-

The normal response returns the slave address, function code, starting address, and quantity of registers written to.

Rx Packet (Response)

Field Name	Example (Hex)
Slave Address	11
Function	10
Starting Address Hi Byte	00
Starting Address Lo Byte	01
# of Registers Hi Byte	00
# of Registers Lo Byte	02
Error Check (CRC)	-

Exception Responses

For a given entry in a GWT data view, the Exceptions field lists any exception responses returned for a particular request to a Modbus slave device. Exception codes may be returned when there is a problem with the request that was received by the Modbus device. There are 8 pre-defined exception response codes in the Modbus specification:

Code	Name	Description
01	ILLEGAL FUNCTION	The function code received in the query is not an allowable action for the slave.
02	ILLEGAL DATA ADDRESS	The data address received in the query is not an allowable address for the slave.
03	ILLEGAL DATA VALUE	A value contained in the query data field is not an allowable value for the slave
04	SLAVE DEVICE FAILURE	An Unrecoverable error occurred while the slave was attempting to perform the requested action.
05	ACKNOWLEDGE	The slave has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the master.
06	SLAVE DEVICE BUSY	The slave is engaged in processing a long-duration program command. The master should retransmit the message later when the slave is free.
07	NEGATIVE ACKNOWLEDGE	The slave cannot perform the program function received in the query. This code is returned for an unsuccessful programming request using function code 13 or 14 decimal. The master should request diagnostic or error information from the slave.
08	MEMORY PARITY ERROR	The slave attempted to read extended memory, but detected a parity error in the memory. The master can retry the request, but service may be required on the slave device.

In addition to the pre-defined error codes listed above, the Modbus specification allows custom defined error codes. Refer to documentation for the specific Modbus slave device for an explanation of any custom error codes returned from a particular device.

Exception responses are identified by the value of the second byte in the data packet. In the normal response this byte echoes back the function code of the query packet. In an exception response this byte also has it's most significant bit set. This makes the byte value exactly 80 hexadecimal greater than in the normal response.

The following example shows an Illegal Data Address exception response to a read coil status (function 01) query packet sent from a Modbus slave device with an address of 10.

Rx Packet (Exception Response)

Field Name	Example (Hex)
Slave Address	0A
Function	81
Exception Code	02
Error Check (CRC)	-

Document Control

Document Title: **V3.30 Release 3 Modbus Gateway User Manual**

Document Filename: **V3.30 Release 3 Modbus Gateway User Manual.pdf**

Revision	Changes
0.1	Preliminary Version Created for Version 3.30 Release 2 Modbus Gateway
1.0	First edition coinciding with Version 3.30 Release 2 Modbus Gateway
2.0	Second edition coinciding with Version 3.30 Release 3 Modbus Gateway